Exhibit 13 to Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

Example Southwest Count 6 Systems and Services
U.S. Patent No. 7,257,582 ("'582 Patent")

The Accused Systems and Services include without limitation Southwest systems and services that utilize Hadoop; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future Southwest systems and services that have the same or substantially similar features as the specifically identified systems and services ("Example Southwest Count 6 Systems and Services" or "Southwest Systems and Services").[1]

On information and belief, the Southwest Systems and Services use Hadoop in its private cloud(s). For example, Southwest posts, or has posted, job opportunities that require familiarity with Hadoop concepts.

See https://www.linkedin.com/in/meghanaa-malleboina/, job profile of data engineer who stated that they enhanced Hadoop YARN cluster efficiency by 35% through optimized resource allocation.

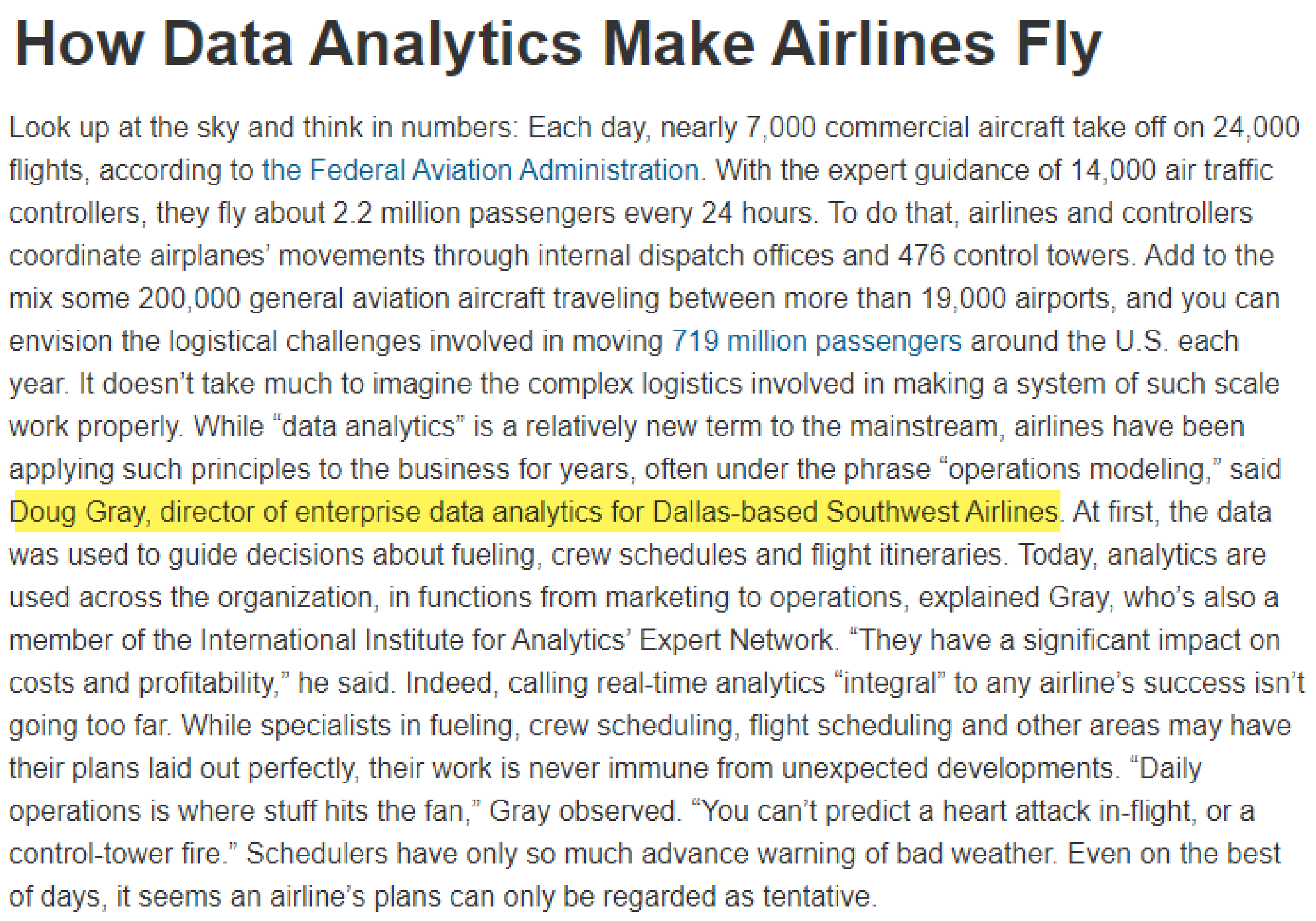
See https://www.linkedin.com/in/rohan-kumar-9861bb127, job profile of Senior Hadoop Developer who stated that they are/were a Hadoop developer for Southwest Airlines. (last accessed 10/8/24).

See https://www.linkedin.com/in/harish-t-b82094128, job profile of Senior Hadoop Developer who stated that they are/were a Hadoop developer for Southwest Airlines. (last accessed 10/8/24).

As another example, Southwest has stated that it is investing in cloud technology and has "moved about 50% of its technology" to the cloud and has indicated cloud migration is one of its areas of focus for 2024 and beyond. Source: https://www.phocuswire.com/southwest-airlines-cio-tech-investment.

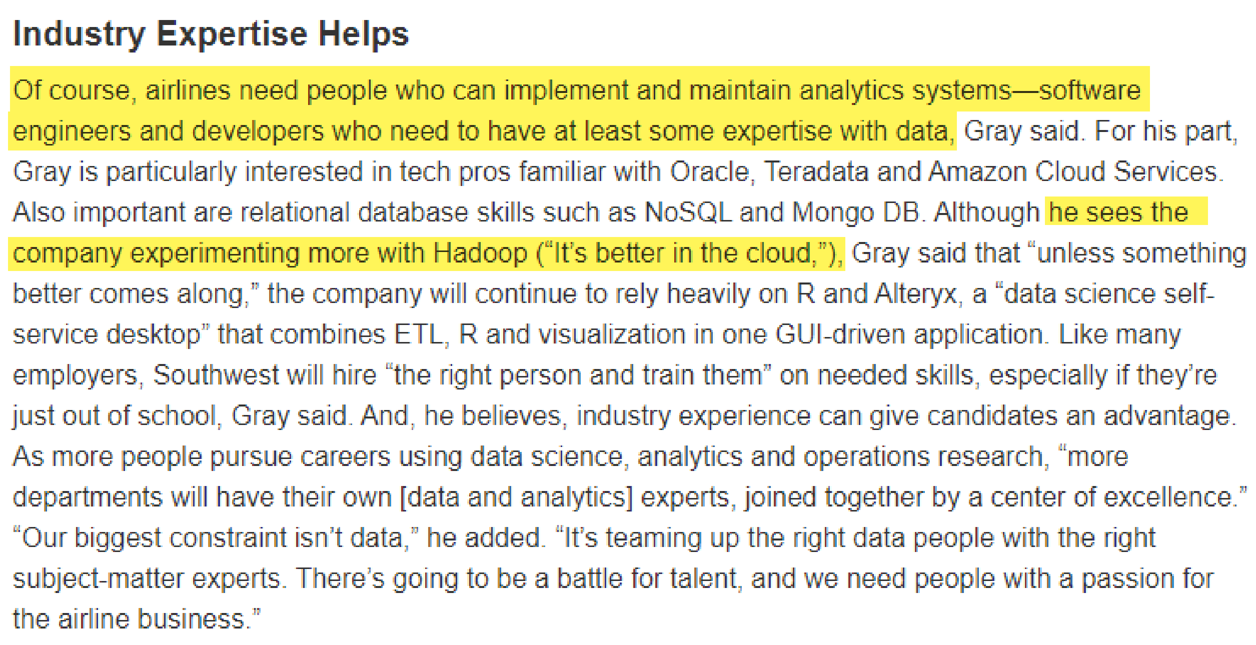
[1] For the avoidance of doubt, Plaintiffs do not accuse public clouds of Southwest if those services are provided by a cloud provider with a license to Plaintiffs' patents that covers Southwest's activities. IV will provide relevant license agreements for cloud providers in discovery. To the extent any of these licenses are relevant to Southwest's activities, Plaintiffs will meet and confer with Southwest about the impact of such license(s).

On information and belief, other information confirms Southwest uses Hadoop technology.



How Data Analytics Make Airlines Fly

Look up at the sky and think in numbers: Each day, nearly 7,000 commercial aircraft take off on 24,000 flights, according to the Federal Aviation Administration. With the expert guidance of 14,000 air traffic controllers, they fly about 2.2 million passengers every 24 hours. To do that, airlines and controllers coordinate airplanes' movements through internal dispatch offices and 476 control towers. Add to the mix some 200,000 general aviation aircraft traveling between more than 19,000 airports, and you can envision the logistical challenges involved in moving 719 million passengers around the U.S. each year. It doesn't take much to imagine the complex logistics involved in making a system of such scale work properly. While "data analytics" is a relatively new term to the mainstream, airlines have been applying such principles to the business for years, often under the phrase "operations modeling," said Doug Gray, director of enterprise data analytics for Dallas-based Southwest Airlines. At first, the data was used to guide decisions about fueling, crew schedules and flight itineraries. Today, analytics are used across the organization, in functions from marketing to operations, explained Gray, who's also a member of the International Institute for Analytics' Expert Network. "They have a significant impact on costs and profitability," he said. Indeed, calling real-time analytics "integral" to any airline's success isn't going too far. While specialists in fueling, crew scheduling, flight scheduling and other areas may have their plans laid out perfectly, their work is never immune from unexpected developments. "Daily operations is where stuff hits the fan," Gray observed. "You can't predict a heart attack in-flight, or a control-tower fire." Schedulers have only so much advance warning of bad weather. Even on the best of days, it seems an airline's plans can only be regarded as tentative.

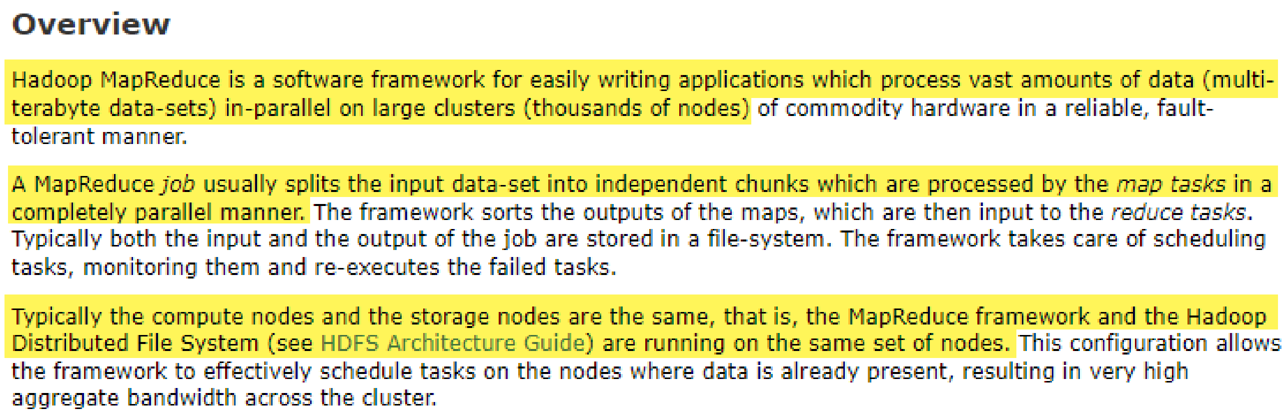


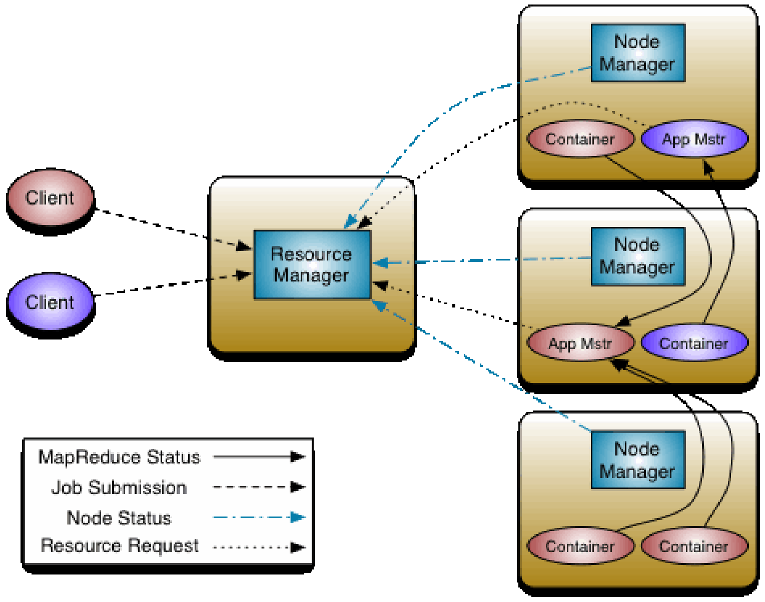
Industry Expertise Helps

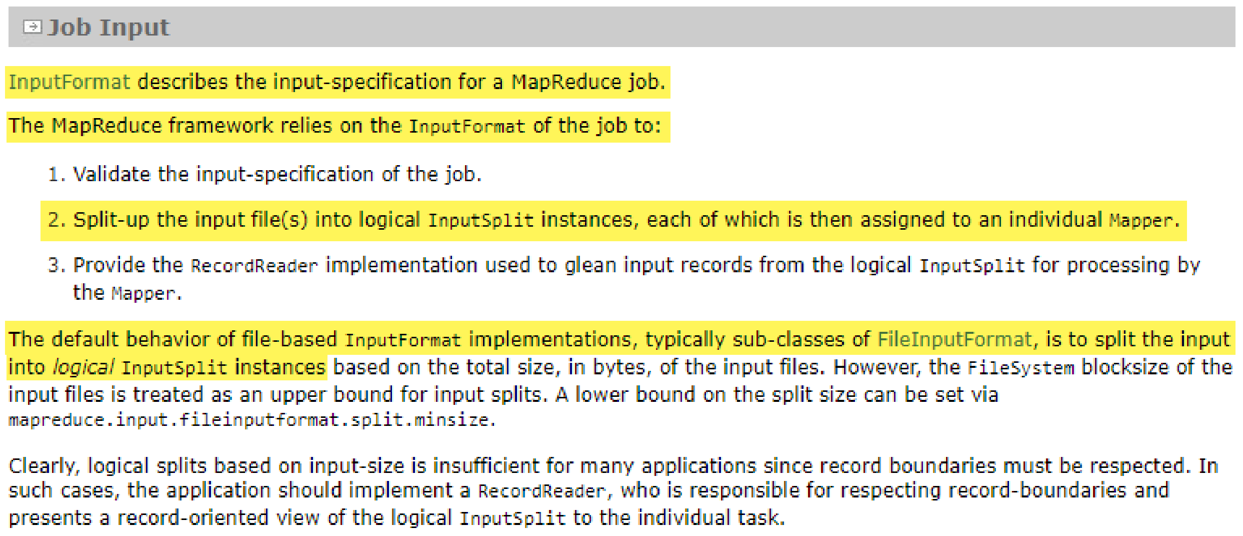
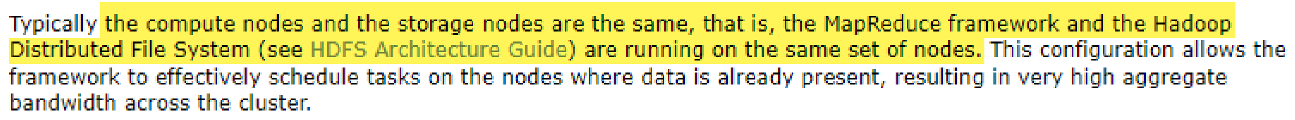
Of course, airlines need people who can implement and maintain analytics systems—software engineers and developers who need to have at least some expertise with data, Gray said. For his part, Gray is particularly interested in tech pros familiar with Oracle, Teradata and Amazon Cloud Services. Also important are relational database skills such as NoSQL and Mongo DB. Although he sees the company experimenting more with Hadoop ("It's better in the cloud,"), Gray said that "unless something better comes along," the company will continue to rely heavily on R and Alteryx, a "data science self-service desktop" that combines ETL, R and visualization in one GUI-driven application. Like many employers, Southwest will hire "the right person and train them" on needed skills, especially if they're just out of school, Gray said. And, he believes, industry experience can give candidates an advantage. As more people pursue careers using data science, analytics and operations research, "more departments will have their own [data and analytics] experts, joined together by a center of excellence." "Our biggest constraint isn't data," he added. "It's teaming up the right data people with the right subject-matter experts. There's going to be a battle for talent, and we need people with a passion for the airline business."

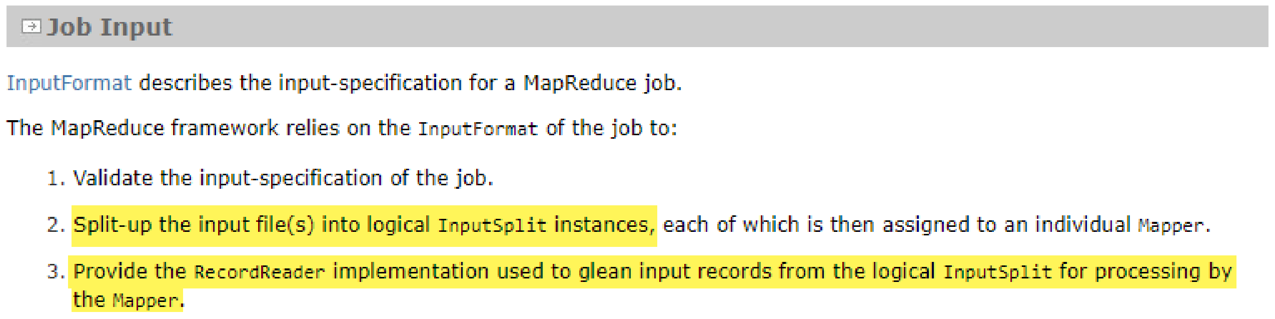
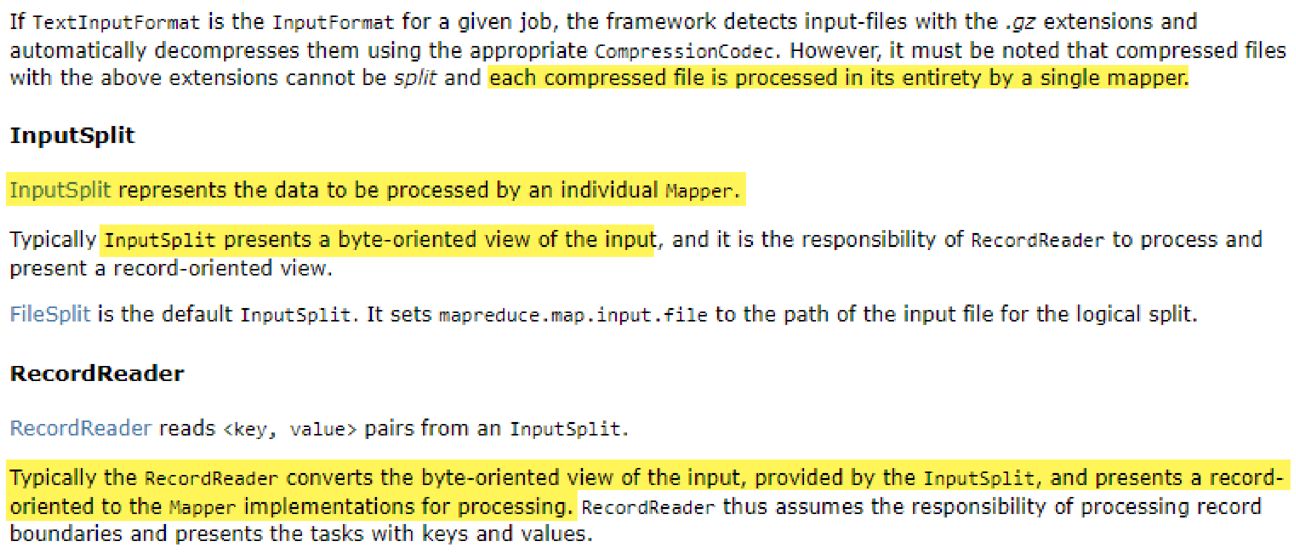
Source: https://www.dice.com/career-advice/how-data-analytics-airlines-fly. [2]

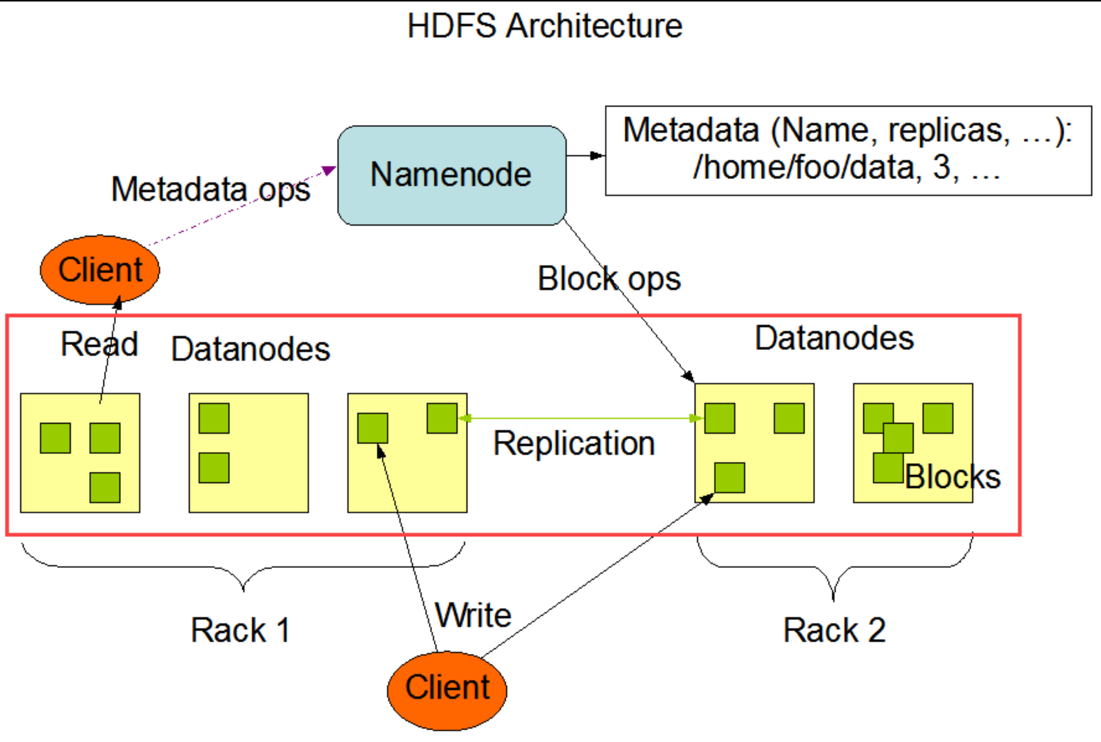
[2] All sources cited in this document were publicly accessible as of the filing date of the Complaint.

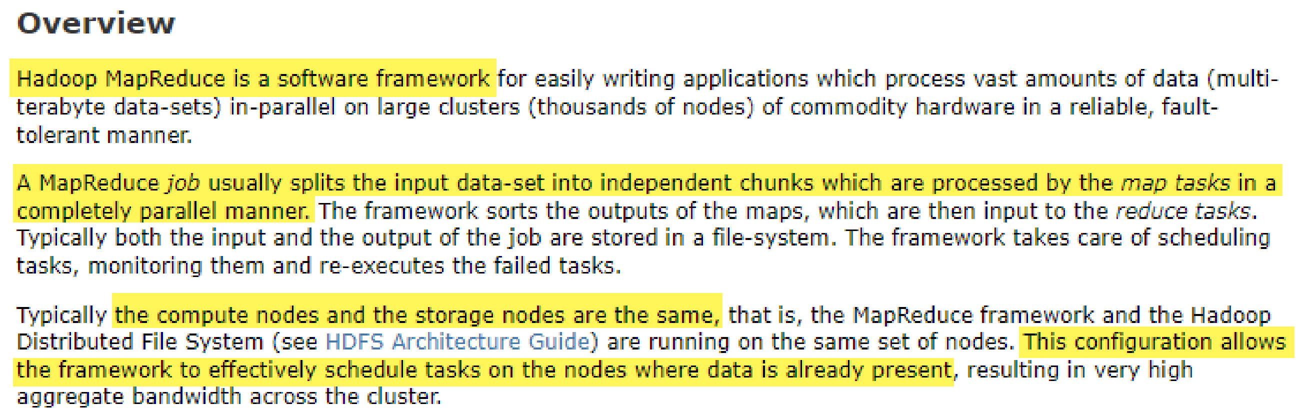
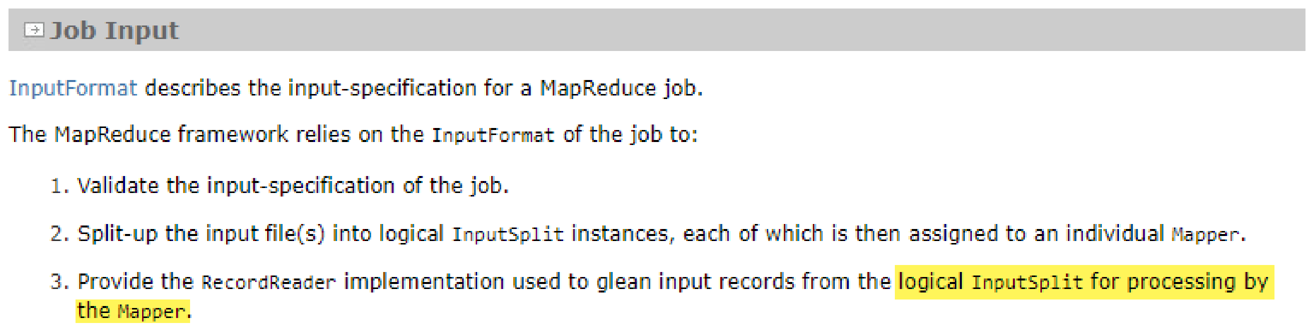
U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
1. A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:	To the extent this preamble is limiting, on information and belief, the Southwest Count 6 Systems and Services practice a method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks. On information and belief, Hadoop includes MapReduce, a software framework that processes multiple map tasks in parallel. An input data set is split into multiple chunks and each chunk is processed by a map task.  **Overview** Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

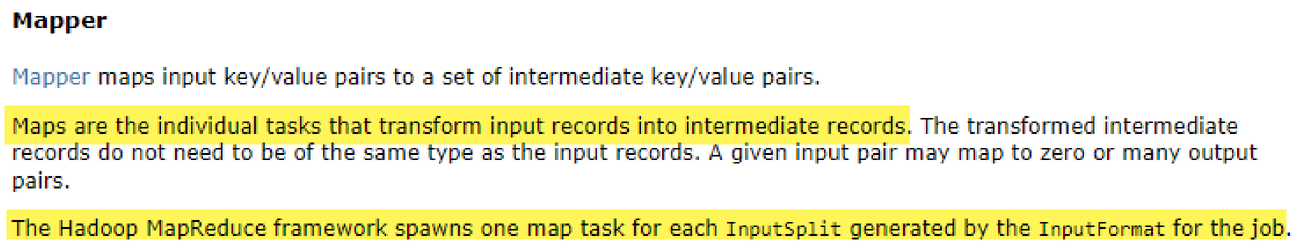
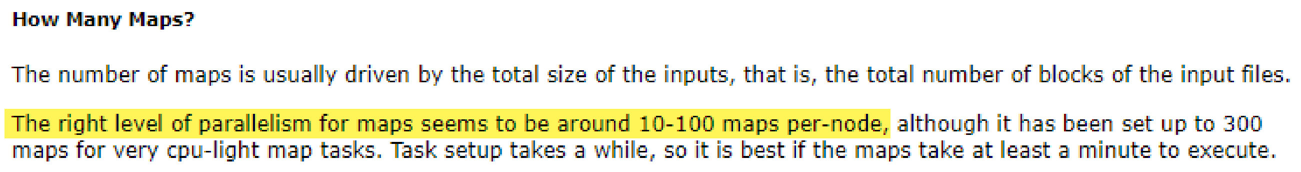
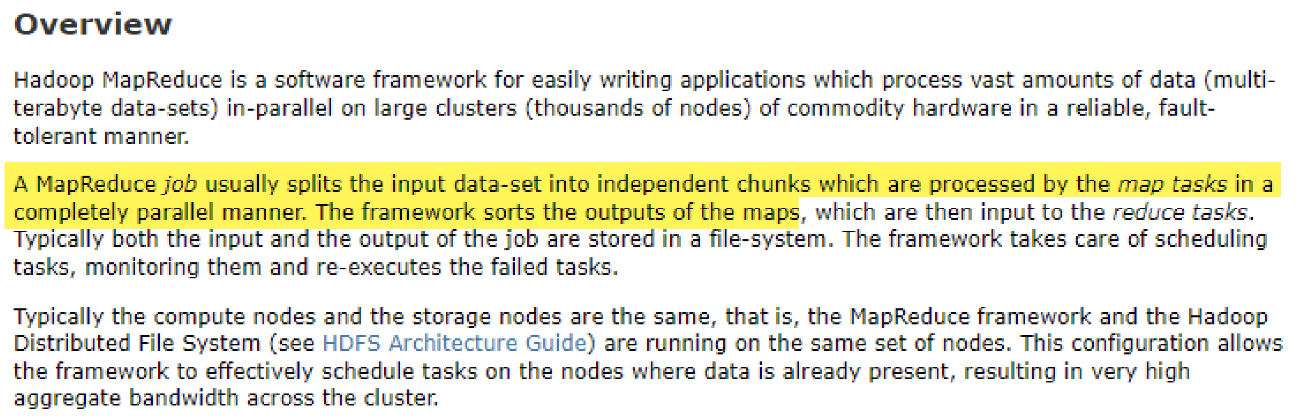
U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	 Client Client Resource Manager Node Manager Container App Mstr Node Manager App Mstr Container Node Manager Container Container MapReduce Status Job Submission Node Status Resource Request Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
(a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;	On information and belief, the Southwest Count 6 Systems and Services practice automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions. On information and belief, Hadoop's InputFormat describes the input-specification for a MapReduce job. The input file is logically split into multiple InputSplit instances.

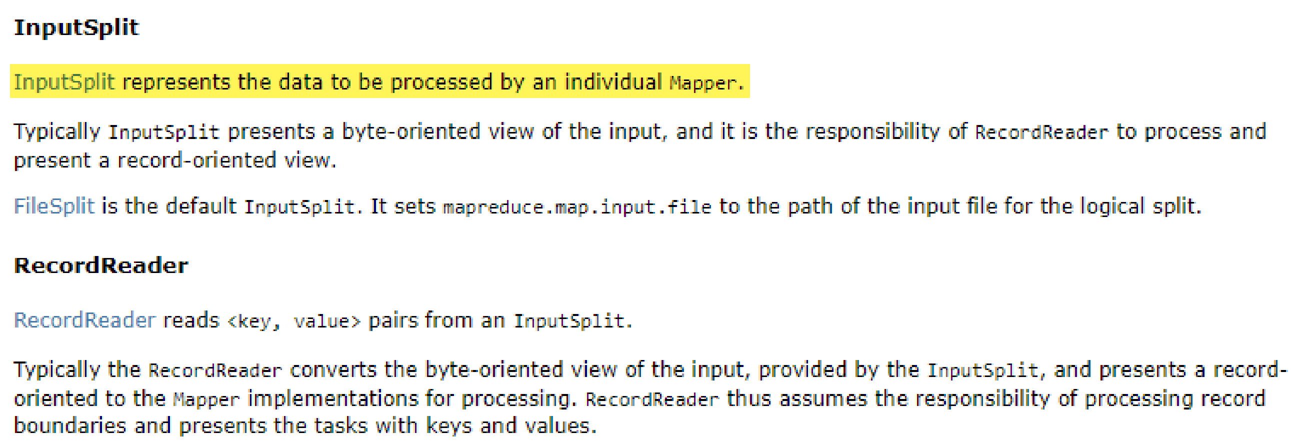
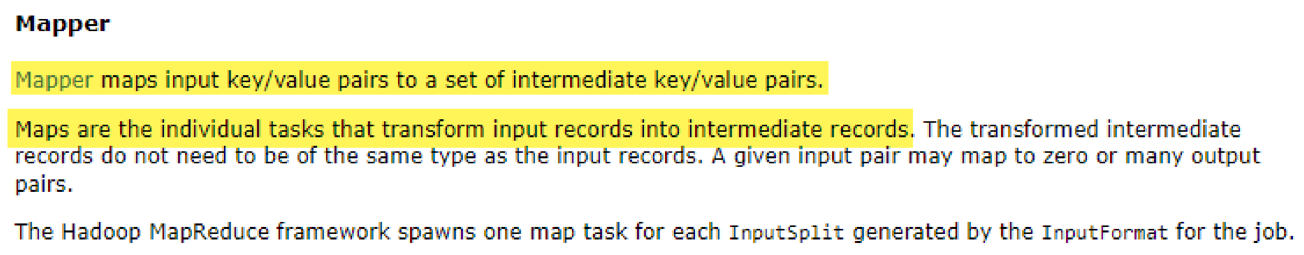
U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	 **Job Input** InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the `InputFormat` of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical `InputSplit` instances, each of which is then assigned to an individual `Mapper`. 3. Provide the `RecordReader` implementation used to glean input records from the logical `InputSplit` for processing by the `Mapper`. The default behavior of file-based `InputFormat` implementations, typically sub-classes of FileInputFormat, is to split the input into *logical* `InputSplit` instances based on the total size, in bytes, of the input files. However, the `FileSystem` blocksize of the input files is treated as an upper bound for input splits. A lower bound on the split size can be set via `mapreduce.input.fileinputformat.split.minsize`. Clearly, logical splits based on input-size is insufficient for many applications since record boundaries must be respected. In such cases, the application should implement a `RecordReader`, who is responsible for respecting record-boundaries and presents a record-oriented view of the logical `InputSplit` to the individual task. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
(b) distributing descriptions of all of said partitions to each of a plurality of subtask processors	On information and belief, the Southwest Count 6 Systems and Services practice distributing descriptions of all of said partitions to each of a plurality of subtask processors. On information and belief, Hadoop's InputSplit represents data that presents a byte oriented view of the input. A recordreader converts the byte oriented view of the data to a record oriented view and presents it to a mapper. Mappers run inside a compute node.  Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

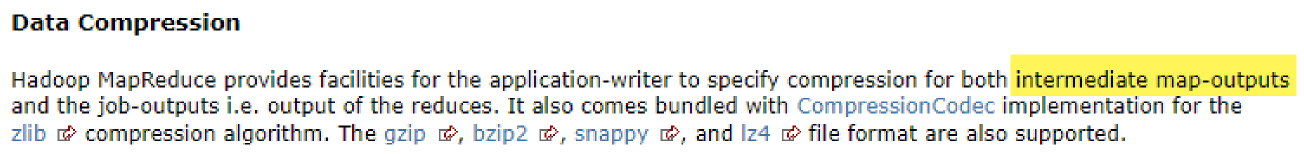
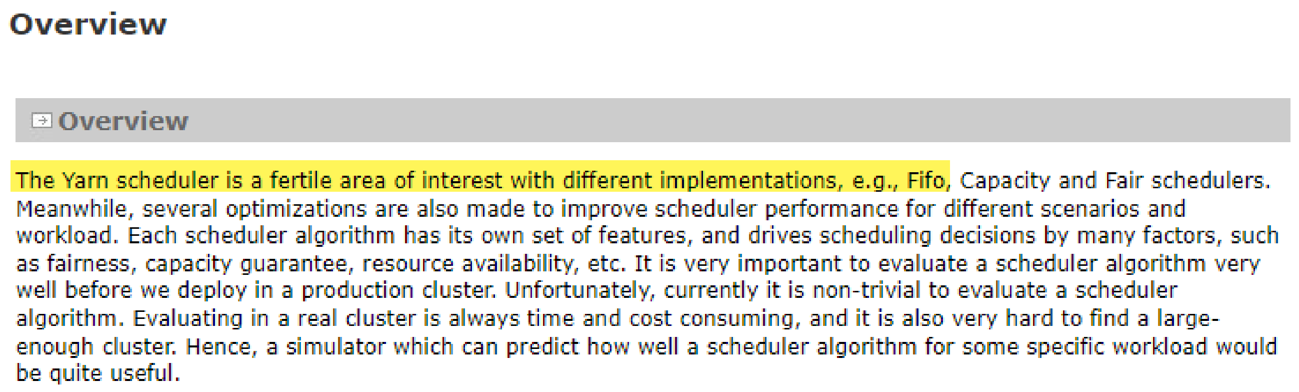
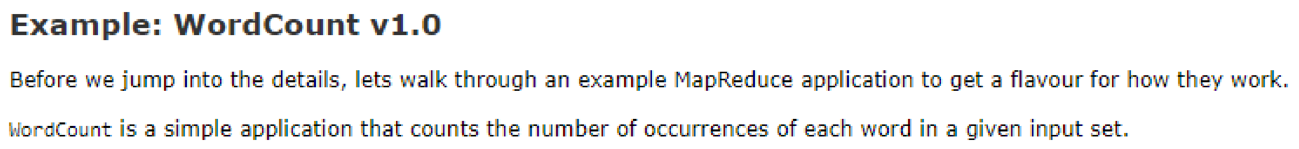
U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	 **Job Input** InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the `InputFormat` of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical `InputSplit` instances, each of which is then assigned to an individual `Mapper`. 3. Provide the `RecordReader` implementation used to glean input records from the logical `InputSplit` for processing by the `Mapper`. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. If `TextInputFormat` is the `InputFormat` for a given job, the framework detects input-files with the *.gz* extensions and automatically decompresses them using the appropriate `CompressionCodec`. However, it must be noted that compressed files with the above extensions cannot be *split* and each compressed file is processed in its entirety by a single mapper. **InputSplit** InputSplit represents the data to be processed by an individual `Mapper`. Typically `InputSplit` presents a byte-oriented view of the input, and it is the responsibility of `RecordReader` to process and present a record-oriented view. FileSplit is the default `InputSplit`. It sets `mapreduce.map.input.file` to the path of the input file for the logical split. **RecordReader** RecordReader reads `<key, value>` pairs from an `InputSplit`. Typically the `RecordReader` converts the byte-oriented view of the input, provided by the `InputSplit`, and presents a record-oriented to the `Mapper` implementations for processing. `RecordReader` thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

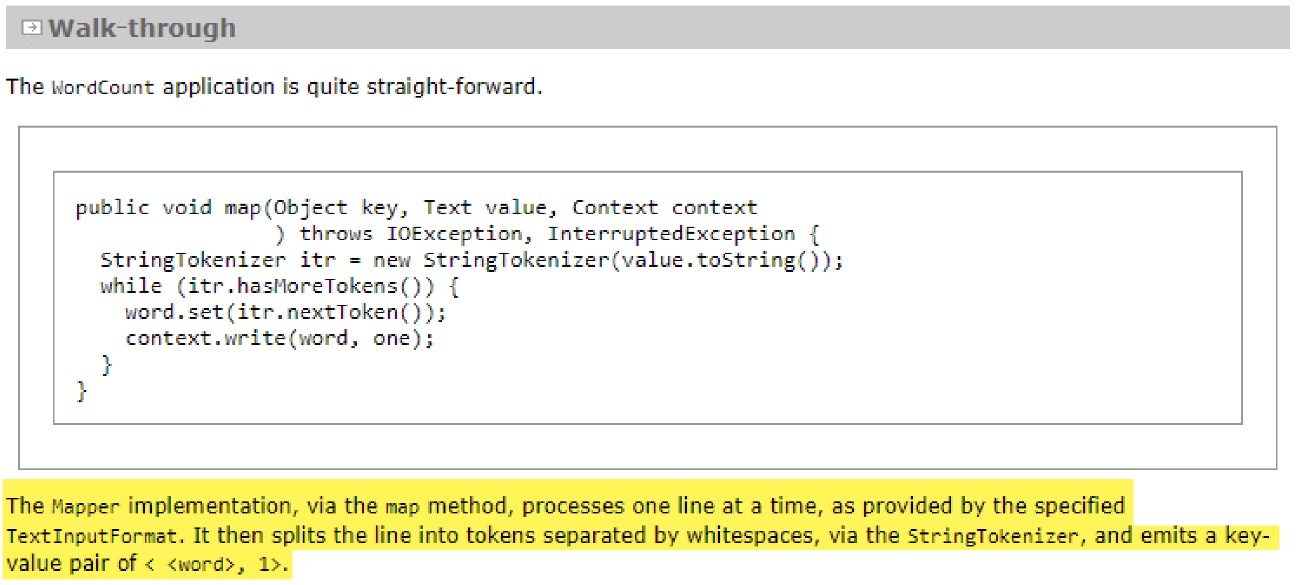
U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	 HDFS Architecture Metadata ops Namenode Metadata (Name, replicas, …): /home/foo/data, 3, … Client Block ops Read Datanodes Datanodes Replication Blocks Rack 1 Write Rack 2 Client Source: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.
(c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective	On information and belief, the Southwest Count 6 Systems and Services practice simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output. On information and belief, Multiple map tasks are executed in parallel. Each Maptask processes a respective InputSplit.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
partition so as to process the respective partition and produce respective subtask output and;	 **Overview** Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.  **Job Input** InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the `InputFormat` of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical `InputSplit` instances, each of which is then assigned to an individual `Mapper`. 3. Provide the `RecordReader` implementation used to glean input records from the logical `InputSplit` for processing by the `Mapper`. Source: Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	 **Mapper** Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.  **How Many Maps?** The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files. The right level of parallelism for maps seems to be around 10-100 maps per-node, although it has been set up to 300 maps for very cpu-light map tasks. Task setup takes a while, so it is best if the maps take at least a minute to execute. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, Each MapTask generates an intermediate map output.  **Overview** Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.  **InputSplit** InputSplit represents the data to be processed by an individual `Mapper`. Typically `InputSplit` presents a byte-oriented view of the input, and it is the responsibility of `RecordReader` to process and present a record-oriented view. FileSplit is the default `InputSplit`. It sets `mapreduce.map.input.file` to the path of the input file for the logical split. **RecordReader** RecordReader reads `<key, value>` pairs from an `InputSplit`. Typically the `RecordReader` converts the byte-oriented view of the input, provided by the `InputSplit`, and presents a record-oriented to the `Mapper` implementations for processing. `RecordReader` thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.  **Mapper** Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	 **Data Compression** Hadoop MapReduce provides facilities for the application-writer to specify compression for both intermediate map-outputs and the job-outputs i.e. output of the reduces. It also comes bundled with CompressionCodec implementation for the zlib compression algorithm. The gzip, bzip2, snappy, and lz4 file format are also supported. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
(d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and	On information and belief, the Southwest Count 6 Systems and Services practice repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis. On information and belief, Hadoop includes a FIFO scheduler. Each mapper processes an InputSplit at a time. For example, a mapper processes one line of the word count application at a time.  **Overview** **Overview** The Yarn scheduler is a fertile area of interest with different implementations, e.g., Fifo, Capacity and Fair schedulers. Meanwhile, several optimizations are also made to improve scheduler performance for different scenarios and workload. Each scheduler algorithm has its own set of features, and drives scheduling decisions by many factors, such as fairness, capacity guarantee, resource availability, etc. It is very important to evaluate a scheduler algorithm very well before we deploy in a production cluster. Unfortunately, currently it is non-trivial to evaluate a scheduler algorithm. Evaluating in a real cluster is always time and cost consuming, and it is also very hard to find a large-enough cluster. Hence, a simulator which can predict how well a scheduler algorithm for some specific workload would be quite useful. Source: https://hadoop.apache.org/docs/r2.9.2/hadoop-sls/SchedulerLoadSimulator.html.  **Example: WordCount v1.0** Before we jump into the details, lets walk through an example MapReduce application to get a flavour for how they work. `WordCount` is a simple application that counts the number of occurrences of each word in a given input set.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. 

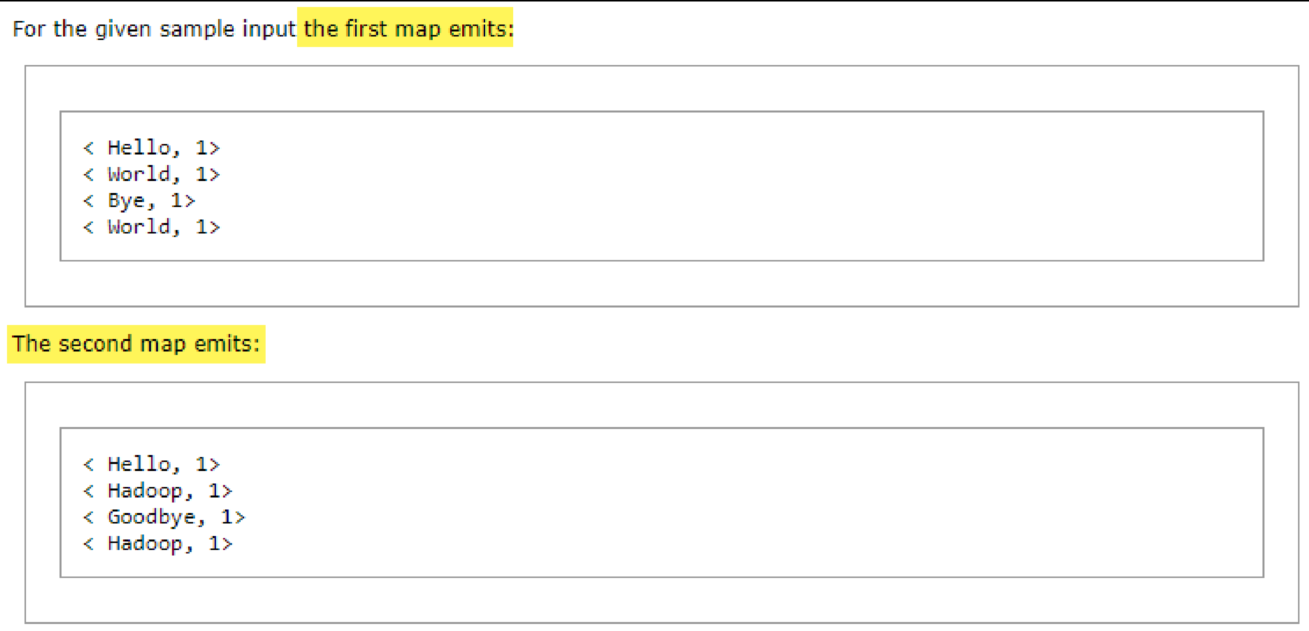
Walk-through

The `WordCount` application is quite straight-forward.

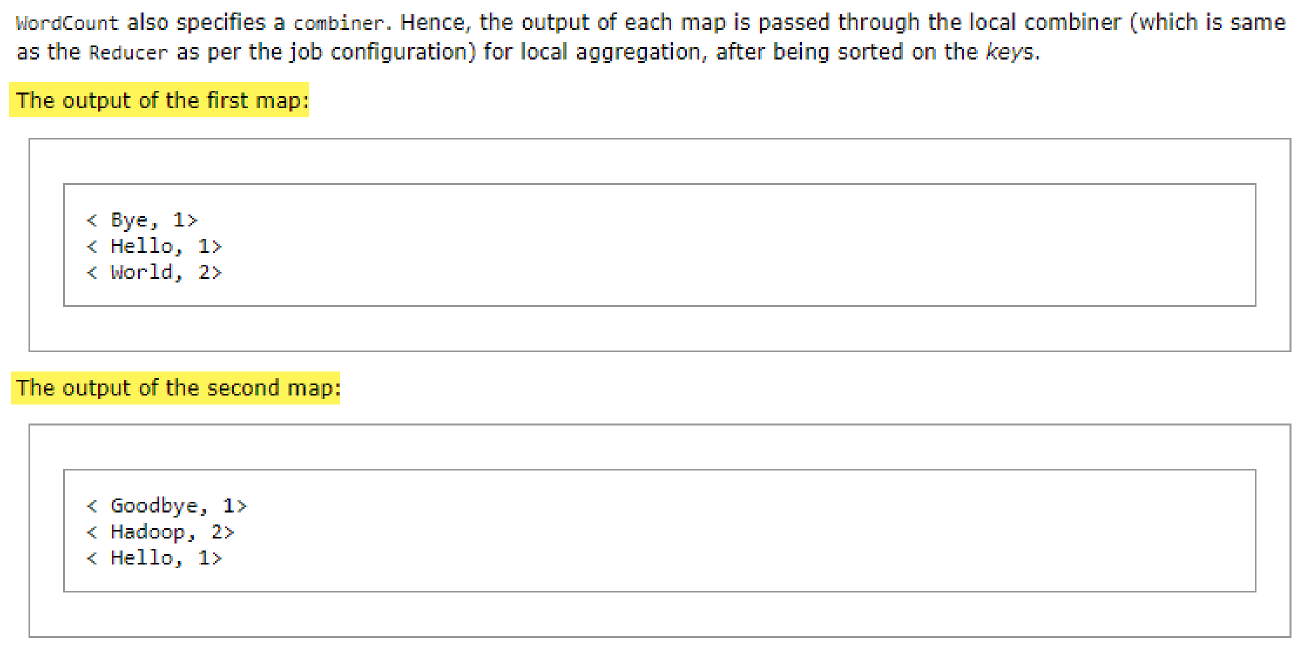
```
public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
  StringTokenizer itr = new StringTokenizer(value.toString());
  while (itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    context.write(word, one);
  }
}
```

The `Mapper` implementation, via the `map` method, processes one line at a time, as provided by the specified `TextInputFormat`. It then splits the line into tokens separated by whitespaces, via the `StringTokenizer`, and emits a key-value pair of `< <word>, 1>`.

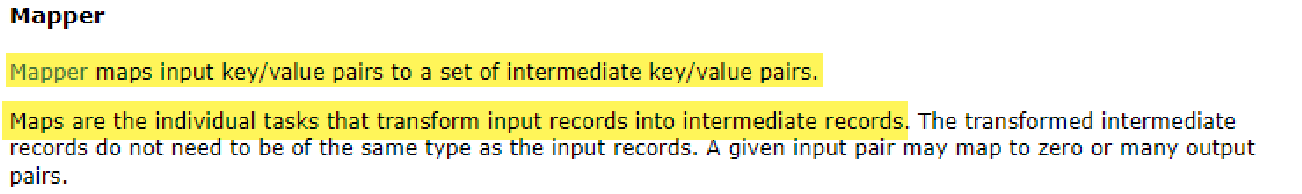
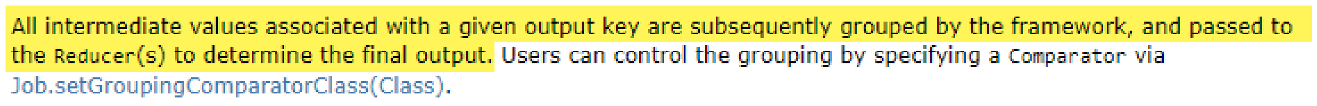
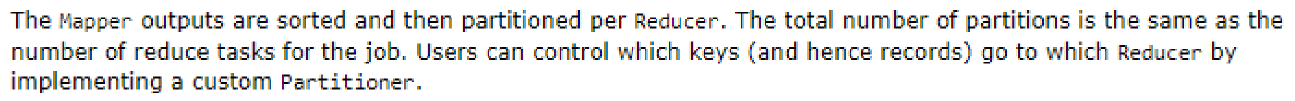
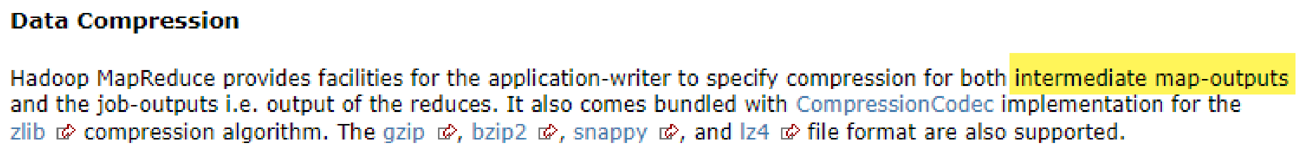
Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	 For the given sample input the first map emits: `< Hello, 1>` `< World, 1>` `< Bye, 1>` `< World, 1>` The second map emits: `< Hello, 1>` `< Hadoop, 1>` `< Goodbye, 1>` `< Hadoop, 1>` Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

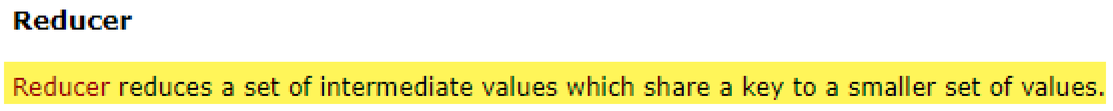
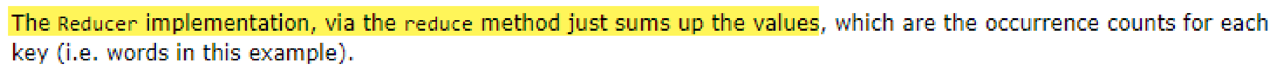
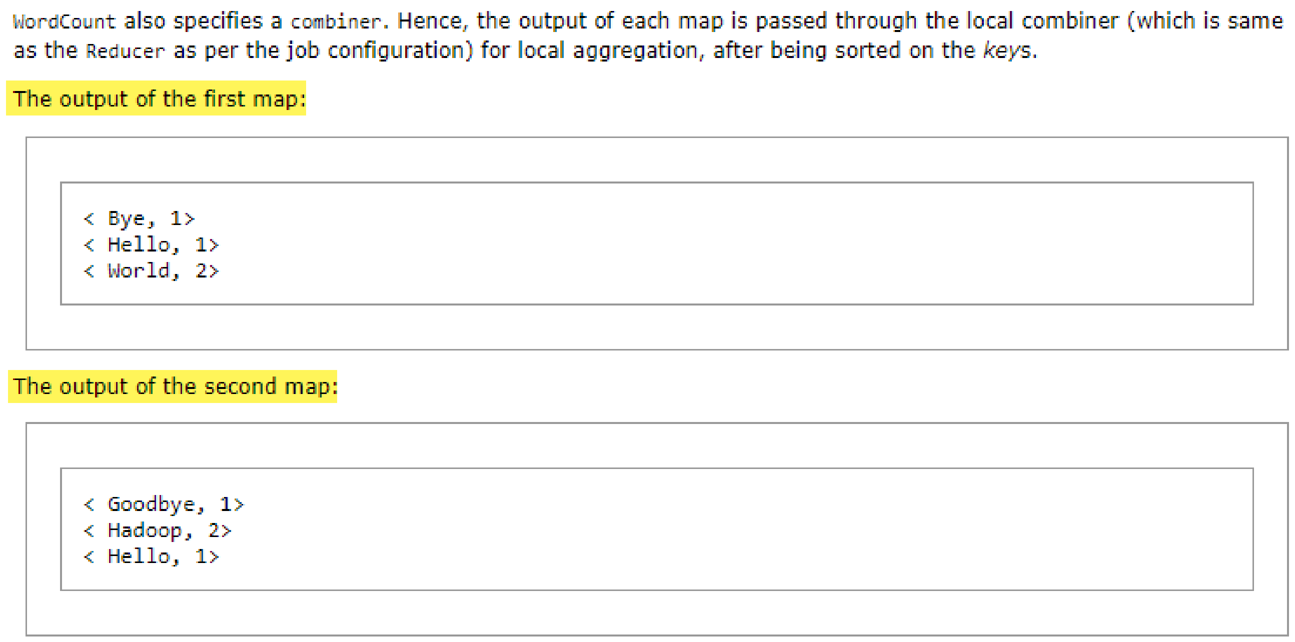
U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**

	 `WordCount` also specifies a `combiner`. Hence, the output of each map is passed through the local combiner (which is same as the `Reducer` as per the job configuration) for local aggregation, after being sorted on the *keys*.
	
	The output of the first map:
	`< Bye, 1>`
	`< Hello, 1>`
	`< World, 2>`
	
	The output of the second map:
	`< Goodbye, 1>`
	`< Hadoop, 2>`
	`< Hello, 1>`
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

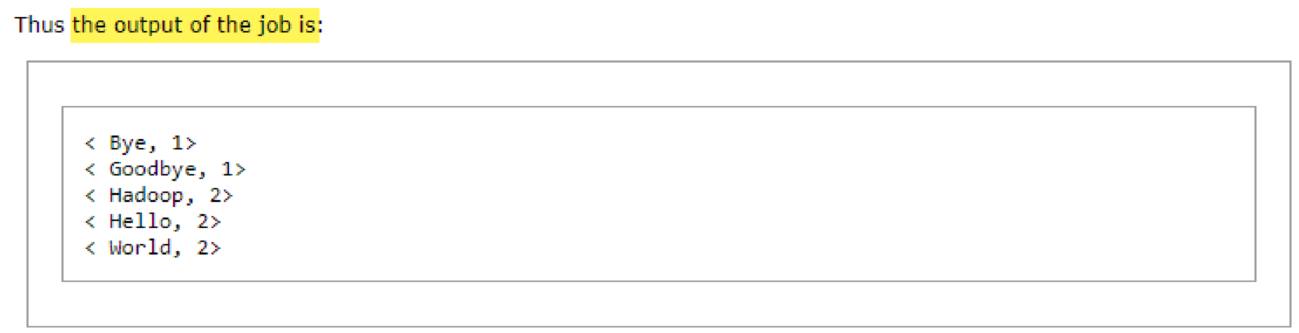
(e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.	On information and belief, the Southwest Count 6 Systems and Services practice generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks. On information and belief, intermediate values are grouped by the Hadoop framework and passed to a Reducer to determine the final output.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**
	 **Mapper** Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.  All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the `Reducer`(s) to determine the final output. Users can control the grouping by specifying a `Comparator` via Job.setGroupingComparatorClass(Class). Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.  The `Mapper` outputs are sorted and then partitioned per `Reducer`. The total number of partitions is the same as the number of reduce tasks for the job. Users can control which keys (and hence records) go to which `Reducer` by implementing a custom `Partitioner`. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.  **Data Compression** Hadoop MapReduce provides facilities for the application-writer to specify compression for both intermediate map-outputs and the job-outputs i.e. output of the reduces. It also comes bundled with CompressionCodec implementation for the zlib compression algorithm. The gzip, bzip2, snappy, and lz4 file format are also supported. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, the Reducer reduces a set of intermediate values to a smaller output.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**

	 **Reducer**
	
	Reducer reduces a set of intermediate values which share a key to a smaller set of values.
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
	
	 The `Reducer` implementation, via the `reduce` method just sums up the values, which are the occurrence counts for each key (i.e. words in this example).
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
	
	 `WordCount` also specifies a `combiner`. Hence, the output of each map is passed through the local combiner (which is same as the `Reducer` as per the job configuration) for local aggregation, after being sorted on the *keys*.
	
	The output of the first map:
	`< Bye, 1>`
	`< Hello, 1>`
	`< World, 2>`
	
	The output of the second map:
	`< Goodbye, 1>`
	`< Hadoop, 2>`
	`< Hello, 1>`
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example Southwest Count 6 Systems and Services**

	core/MapReduceTutorial.html.
	
	Thus the output of the job is:
	`< Bye, 1>`
	`< Goodbye, 1>`
	`< Hadoop, 2>`
	`< Hello, 2>`
	`< World, 2>`
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.